smart_model_training

October 16, 2020

1 S.Ma.R.T. MODEL TRAINING

[1]:
```
# Se define el estilo de visualización del notebook:
from IPython.core.display import display, HTML
display(HTML("<style>.container { width:90% !important; }</style>"))
```

```
<IPython.core.display.HTML object>
```

[2]:
```
import time
from datetime import timedelta
start_time = time.time()
```

[3]:
```
import datetime as dt
import numpy    as np
import pandas   as pd
import scipy

extrae_id = lambda x : (x.strip('ObjectId').strip('("')'))

def remove_outliers_6sigma(X):
    return X[abs(X - np.mean(X)) < 3 * np.std(X)]
```

[4]:
```
ruta = "./dataset/all_data.csv"
df   = pd.read_csv(ruta, converters = {'_id':extrae_id , 'DrillID':extrae_id})
```

[5]:
```
df.sort_values(['TagID', 'DrillID', 'TimeStamp'], ascending=[True, True, True],␣
↪inplace=True)
```

[6]:
```
if df.duplicated().sum() > 0:
    df = df.drop_duplicates()
```

[7]:
```
df.drop(list(df.columns[df.apply(pd.Series.nunique)==1]), axis=1, inplace=True)
```

[8]:
```
if df.isnull().sum().sum() > 0:
    df = df.fillna(0)
```

[9]:
```
df['PositionX'] = pd.to_numeric(df['Position'].str.strip('[]').str.
↪split(pat="," , expand = True)[0], downcast='float')
```

[10]:
```
df.drop(['Position'], axis=1, inplace=True)
```

[11]:
```
df['TimeStampDT'] = pd.to_datetime(df['TimeStamp'], unit='ms')
```

[12]:
```
df  = df[['_id', 'TagID', 'DrillID', 'TimeStamp', 'TimeStampDT', 'PositionX']]
```

[13]:
```
df["diff_posx_m"]   = df["PositionX"].diff().abs()
df["diff_time_sec"] = df["TimeStamp"].diff().abs() / 1000
df["LinearSpeed"]   = df["diff_posx_m"].div(df["diff_time_sec"])
df["Acceleration"]  = df["diff_posx_m"].div(df["diff_time_sec"]**2)
```

[14]:
```
if df.isnull().sum().sum() > 0:
    df = df.fillna(0)
```

[15]:
```
df = df.drop( (df[df['LinearSpeed']  > 1.56].index ) )
df = df.drop( (df[df['Acceleration'] > 0.03].index ) )
```

[16]:
```
variables = ["diff_posx_m", "diff_time_sec", "LinearSpeed", "Acceleration"]
```

[ ]:
```
df_aux1     = pd.merge(df     , pd.
↪DataFrame(remove_outliers_6sigma(df[variables[0]].values)), how = 'left',␣
↪left_index = True,
                       right_index = True).rename(columns={0: variables[0] +␣
↪'_no_outlier'})
df_aux2     = pd.merge(df_aux1, pd.
↪DataFrame(remove_outliers_6sigma(df[variables[1]].values)), how = 'left',␣
↪left_index = True,
                       right_index = True).rename(columns={0: variables[1] +␣
↪'_no_outlier'})
df_aux3     = pd.merge(df_aux2, pd.
↪DataFrame(remove_outliers_6sigma(df[variables[2]].values)), how = 'left',␣
↪left_index = True,
                       right_index = True).rename(columns={0: variables[2] +␣
↪'_no_outlier'})
df_filtered = pd.merge(df_aux3, pd.
↪DataFrame(remove_outliers_6sigma(df[variables[3]].values)), how = 'left',␣
↪left_index = True,
                       right_index = True).rename(columns={0: variables[3] +␣
↪'_no_outlier'})

df_filtered.dropna(inplace=True)
```

```
[ ]: df_filtered.drop(variables, axis=1, inplace=True)
     df_filtered.rename(columns={"diff_posx_m_no_outlier"  : "diff_pos",
                                 "diff_time_sec_no_outlier": "diff_time",
                                 "LinearSpeed_no_outlier"  : "Speed",
                                 "Acceleration_no_outlier" : "Accel"}, inplace=True)

[ ]: df  = df_filtered.copy()

[ ]: df_agg = df.groupby(['TagID', 'DrillID']).agg({'TimeStamp'  : ['min','max'],
                                                    'TimeStampDT': ['min','max'],
                                                    'PositionX'  : ['min','max',␣
      ↪'mean'],
                                                    'diff_pos'   : ['min','max',␣
      ↪'mean'],
                                                    'diff_time'  : ['min','max',␣
      ↪'mean'],
                                                    'Speed'      : ['min','max',␣
      ↪'mean'],
                                                    'Accel'      : ['min','max',␣
      ↪'mean']}).reset_index()

[ ]: df6 = df_agg.copy()

[ ]: df_agg = df_agg.drop(['TagID', 'DrillID', 'TimeStamp', 'TimeStampDT'], axis = 1)

[ ]: df_agg['Outlier_PositionX'] = ( df_agg['PositionX']['max'] *␣
      ↪df_agg['PositionX']['min'] ) >= 0

[ ]: df_mask_X  = df_agg['Outlier_PositionX']==False
     filtered_X = df_agg[df_mask_X]
     df_agg     = filtered_X.drop('Outlier_PositionX', axis=1)

[ ]: df_agg.columns = ['_'.join(col_).strip() for col_ in df_agg.columns.values]

[ ]: df_ul = df_agg.copy()

[ ]: df_ul['classifier_'] = np.log( (1 + df_ul['Speed_mean']) / ( 1 +␣
      ↪(df_ul['Accel_mean'] * df_ul['diff_time_mean']) ) )

[ ]: df_ul.drop(['diff_time_max', 'Speed_mean'], axis=1, inplace=True)

[ ]: from sklearn.preprocessing import MinMaxScaler
     min_max           = MinMaxScaler()
     df_scaled         = pd.DataFrame(min_max.fit_transform(df_ul.astype("float64")))
     df_scaled.columns = df_ul.columns
```

[ ]:
```
from sklearn.cluster import KMeans
number_of_clusters = 3
kmeans             = KMeans(n_clusters = number_of_clusters)
kmeans.fit(df_scaled)
```

[ ]:
```
import joblib
joblib.dump(kmeans, 'smart_new_model.pkl', compress=9)
```

[ ]:
```
elapsed_time_secs = time.time() - start_time
msg = "Execution took: %s secs (Wall clock time)" %␣
 ↪timedelta(seconds=round(elapsed_time_secs))
print(msg)
```

[ ]:
```
!jupyter nbconvert --to pdf smart_model_training.ipynb
```

[ ]:
```

```